Représentation compacte des adresses pour le routage par caractéristiques

Michał Król[1], Franck Rousseau[1], Andrzej Duda[1]

[1]*Grenoble INP, LIG, CNRS UMR 5217, Grenoble, France*

Nous nous intéressons à la représentation compacte des adresses afin de pouvoir router des paquets vers des destinations identifiées par leurs caractéristiques. Ce type de routage nécessite le stockage des caractéristiques de manière compacte dans l'adresse destination des paquets et dans les tables de routage. Nous étudions plusieurs solutions dans le cadre des réseaux de capteurs, où les ressources sont très limitées et les adresses suivent le format IPv6.

Keywords: routage, protocoles centrées sur le contenu, filtres de Bloom, réseaux de capteurs

1 Introduction

Le développement de nouveaux paradigmes de routage, comme les approches centrées sur le contenu [HP06, XL06] nécessitent de revoir en profondeur la notion d'adressage et de relayage de paquets. Les adresses ne désignent plus directement des nœuds du réseau, mais des ressources, et il s'agit alors de router des intérêts ou des données vers la ou les bonnes destinations.

Nous considérons une nouvelle approche à base de caractéristiques (*featurecast*) : aux nœuds d'un réseau de capteurs sont associées leurs caractéristiques sous forme de prédicat f_i. Une adresse destination A prend forme alors d'une conjonction de caractéristiques : $A = \{f_1, f_2, ..., f_n\}, f_i \in \mathcal{F}$, par exemple $\{\texttt{lumière}, 3^{\texttt{e}}\ \texttt{étage}\}$, où $\mathcal{F}$ est l'ensemble de toutes les caractéristiques définies dans le réseau. À une adresse A correspond l'ensemble des nœuds du réseau dont A est un sous-ensemble des caractéristiques. Le routage consiste alors à acheminer les paquets d'adresse destination A à tous ces nœuds, *c.à.d.* ceux possédant les caractéristiques présentes dans A, par le mécanisme suivant : les entrées de table de routage sont de la forme $[nh_k, \mathcal{F}_k]$, où $\mathcal{F}_k$ est l'ensemble des caractéristiques annoncées par les nœuds joignables à travers le prochain saut nh_k ; une copie du paquet sera envoyée à nh_k si $A \subseteq \mathcal{F}_k$, *c.à.d.* si les caractéristiques de A sont présentes dans les nœuds joignables via le voisin nh_k.

Afin de simplifier l'utilisation de cet adressage, nous n'imposons aucune contrainte sur la définition des caractéristiques, qui ne sont pas tenues d'être structurées. De plus, afin de pouvoir fonctionner conjointement au routage IPv6 classique, nous prenons comme contrainte de pouvoir représenter les adresses au format *multicast* IPv6, ce qui laisse 112 bits pour le codage.

Les filtres de Bloom [TRL12] sont un moyen efficace de stocker un ensemble d'éléments dans une structure compacte, pour pouvoir ensuite tester l'appartenance à cet ensemble d'un élément quelconque. Ces filtres semblent donc répondre à notre besoin de routage par caractéristiques dans les réseaux de capteurs en nous dispensant d'un coûteux système de traduction des adresses à la DNS. Nous faisons cependant face à deux contraintes contradictoires : (i) nous devons pouvoir encoder un ensemble assez réduit de caractéristiques de manière suffisamment compacte pour coder les adresses destination sur 112 bits afin de minimiser le surcoût de transport, et (ii) nous devons pouvoir encoder un ensemble très grand de caractéristiques, potentiellement $\mathcal{F}$, dans les entrées de table de routage. Dans cet article, nous présentons et analysons plusieurs solutions permettant de résoudre notre problème.

2 Représentation compacte d'éléments

2.1 Filtres de Bloom

Un filtre de Bloom est une structure probabiliste permettant le stockage efficace d'un ensemble d'éléments. Il est constitué d'un vecteur de m bits et d'un ensemble de k fonctions de hachage. Initialement tous les bits sont à 0. Pour insérer un élément dans un filtre, les k fonctions de hachage sont calculées sur celui-ci et leurs résultats déterminent les positions des bits mis à 1. Un bit déjà à 1 ne changera pas. Pour tester si un élément appartient à un filtre, il suffit de calculer ses k fonctions de hachage sur l'élément et vérifier si tous les bits aux positions correspondantes sont à 1. Si ce n'est pas le cas, il est certain que l'élément n'était pas dans filtre.

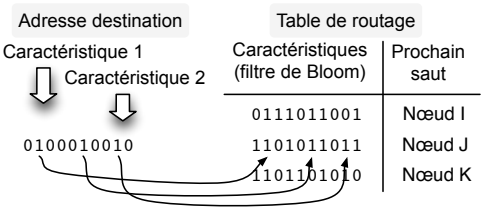
Il reste cependant une probabilité de faux positifs : il est possible que tous les bits correspondant aient été mis à 1 par d'autres éléments stockés, et donc de détecter un élément testé alors que celui-ci n'est pas dans le filtre. La probabilité de faux positifs en fonction du nombre d'éléments stockés n et de la taille du filtre (m, k) est $p \approx (1 - e^{-km/n})^k$. Pour maintenir le même taux de faux positifs avec un nombre croissant d'éléments, il faut augmenter le nombre de bits et de fonctions de hachage, ce qui se traduit par une plus grande consommation de mémoire et une augmentation du coût en calcul.

2.2 Solution 1 : filtres de même taille pour l'adresse destination et dans la table de routage

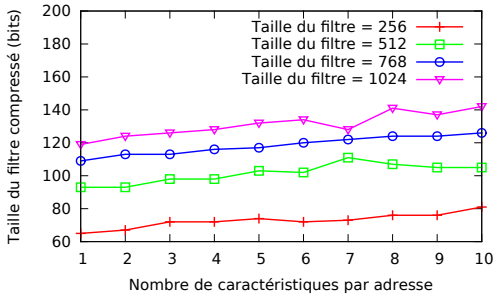
La première solution triviale à notre problème de codage des adresses destination ainsi que des entrées de table de routage consiste à utiliser des filtres de Bloom de taille unique pour stocker les caractéristiques. Une telle solution permet de comparer efficacement l'adresse destination avec les entrées de la table pour décider où envoyer le paquet relayé.

Pour comparer deux filtres, ils doivent avoir la même taille et utiliser le même ensemble de fonctions de hachage, *cf.* FIGURE 1(a). Si l'adresse de destination contient n éléments et p est la probabilité de faux positif dans la tables de routage, alors la probabilité qu'un paquet soit transmis à un voisin qui ne définit pas toutes les caractéristiques nécessaires est égale à $1-(1-p)^n$. Ce taux de faux positifs dépend fortement du nombre d'éléments et de la taille du filtre. Si n est le nombre d'éléments dans le filtre et p est la probabilité requise de faux positifs, le nombre minimum de bits m pour le filtre est donné par : $m \geq n \log_2(e) \log_2(1/p)$.

Pour atteindre une probabilité de faux positifs de 1%, nous avons besoin de 10 bits par élément. Comme nous ne pouvons utiliser que 112 bits d'une adresse multicast IPv6, il est possible de stocker seulement 11 éléments avec cette probabilité de faux positifs. Cette valeur est acceptable pour coder une adresse de destination du paquet, mais insuffisante pour une entrée de table de routage dans laquelle beaucoup plus d'éléments devront apparaitre. Des versions améliorées telles que les filtres de Bloom compressés [Mit02] ne permettent de réduire la taille du filtre que légèrement tout en introduisant un coût supplémentaire en calcul.



(a) Solution 1. Filtres de Bloom de même taille pour l'adresse et la table de routage



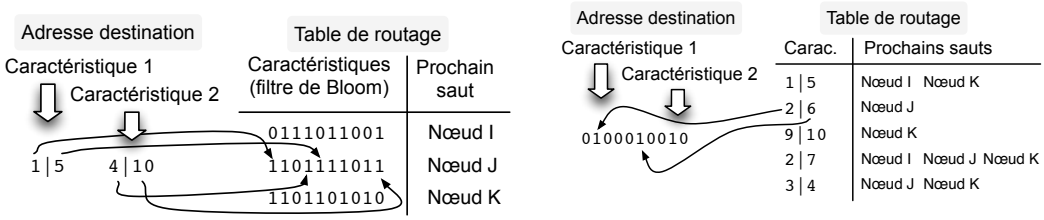
(b) Taille de filtres compressés à l'aide du codage arithmétique adaptatif.

FIGURE 1: Solutions 1 et 2

2.3 Solution 2 : filtres de même taille avec compression

Nous avons besoin de filtres de Bloom beaucoup plus grands dans les tables de routage pour pouvoir stocker de nombreuses caractéristiques. Pour stocker 128 éléments, l'utilisation de 1024 bits conduit à une probabilité de faux positifs $p_{128} \approx 3\%$. Nous n'avons pas 1024 bits disponibles dans le champ d'adresse, cependant, une façon d'utiliser de grands filtres est de les compresser : en effet les filtres d'adresse sont peu denses sachant qu'ils ne contiennent que quelques éléments par rapport à ceux des tables de routage.

Nous avons évalué l'utilisation du codage arithmétique adaptatif (AAC) qui permet d'obtenir des taux de compression proches des bornes théoriques. La FIGURE 1(b) présente la taille du filtre compressé pour différentes tailles de filtre. Cette approche permet donc de compresser un filtre de 768 bits pour l'adapter à un champ d'adresse de 112 bits. Dans ce cas, nous ne pouvons pas représenter 128 éléments, mais au plus 77 avec une probabilité de faux positifs de $\approx 2\%$.



(a) Solution 3. Liste d'éléments hachés pour l'adresse et filtre de Bloom dans la table de routage

(b) Solution 4. Filtre de Bloom pour l'adresse et liste d'éléments hachés dans la table de routage

FIGURE 2: Solutions 3 et 4

2.4 Solution 3 : liste de positions pour l'adresse destination et filtre de Bloom dans la table de routage

Afin de supporter un plus grand nombre d'éléments dans les entrées de la table de routage, nous proposons d'utiliser un filtre de Bloom de taille suffisante pour garantir une faible probabilité de faux positifs. Ce dernier étant de très grande taille, nous ne pouvons pas représenter les adresses de la même façon.

Un élément unique représenté dans le filtre positionne au plus k bits, où k est le nombre de fonctions de hachage. Nous pouvons ainsi représenter uniquement la position de chaque bit dans l'adresse, *cf.* FIGURE 2(a). Sachant les positions des bits à 1, nous pouvons les comparer avec les bits correspondants dans les filtres de la table de routage.

La taille de chaque élément représenté ainsi dépend de la taille du filtre de Bloom et du nombre de fonctions de hachage : $s = k\log_2(m)$. Pour un filtre de Bloom de 1024 bits et 2 fonctions de hachage, $s = 20$, donc on peut représenter jusqu'à 5 éléments dans 112 bits.

2.5 Solution 4 : filtre de Bloom pour l'adresse destination et liste de positions dans la table de routage

Enfin, nous étudions la solution inverse de la précédente : nous utilisons un filtre de Bloom dans le champ d'adresse, et pour supporter un plus grand nombre d'éléments dans le réseau, nous utilisons le codage des positions dans les tables de routage. Un élément dans la table de routage est représenté par les valeurs des fonctions de hachage, *c.à.d.* les positions des bits mis à 1 dans le filtre de Bloom. Par exemple, un élément positionnant les bits 5 et 76, sera représenté par ces deux nombres, *cf.* FIGURE 2(b).

La probabilité que deux éléments différents aient la même représentation est égale à $p_f = f/m^k$, où f est le nombre d'éléments dans le système. Cette approche permet de stocker 200 éléments avec une probabilité de faux positifs inférieure à 2%. Nous avons décidé de stocker une liste d'éléments disponibles pour chacun de nos voisins.

Comme précédemment, la taille de chaque élément est $s = k\log_2(m)$. Le filtre de Bloom est constitué de 2 fonctions de hachage, mais de seulement 112 bits, soit $s = 14$. Nous n'avons besoin que de 2 octets pour stocker un élément dans la table de routage et la taille de celle-ci n'est pas limitée par une borne fixe comme pour l'adresse. Cependant, stocker une liste d'éléments dans la table de routage nécessite de les tester pour savoir s'ils sont dans l'adresse destination du paquet relayé.

TABLE 1: Comparaison des solutions. m = nb. éléments dans l'adresse, n = nb. éléments dans la table de routage.

Solution :	**1. Deux filtres**	**2. Compression**	**3. Filtre dans la table**	**4. Filtre dans l'adresse**
Nb. max d'éléments dans l'adresse	10	10	5	10
Bits/élément dans l'adresse	10	10	18	10
Nb. max d'éléments dans la table	10	102	102	illimité
Bits/élément dans la table	10	10	10	12
Complexité du relayage	$O(1)$	$O(1)$	$O(m)$	$O(n)$

3 Évaluation

Nous avons pris comme contrainte que la solution fonctionne avec IPv6, ainsi, nous avons 112 bits disponibles pour l'adresse. Nous limitons le taux de faux positifs pour chaque solution à 3%. La TABLE 1 montre une comparaison entre les quatre propositions.

Toutes les solutions sauf la 3 permettent de stocker jusqu'à 10 caractéristiques dans l'adresse, ce qui doit être suffisant pour la plupart des réseaux visés. La troisième ligne compare le nombre maximum d'éléments dans la table de routage. Avec le taux de faux positifs maximal donné, le filtre de Bloom simple ne peut supporter que 10 éléments. Ce nombre est insuffisant pour stocker toutes les caractéristiques de nos scénarios, même pour les petits réseaux. Les solutions 2 et 3 permettent de conserver un filtre de 1024 bits dans la table de routage et de prendre en charge jusqu'à 100 éléments. La dernière solution permet de stocker un nombre illimité d'éléments par conception, la limite sera imposée par la mémoire disponibles dans les nœuds capteurs.

Dans la solution 1, la comparaison des paquets est extrêmement efficace et ne dépend ni du nombre d'éléments stockées dans les tables de routage, ni dans l'adresse. La complexité de la version compressée est la même, mais un coût significatif en calcul est induit par le codage et le décodage arithmétique adaptatif des adresses à chaque saut. Dans la solution 3, nous avons besoin de rechercher chaque élément de l'adresse dans la table de routage, ainsi le coût dépend du nombre d'éléments présents dans l'adresse. À l'inverse, pour la solution 4 le coût dépend du nombre d'éléments stockés dans les tables de routage, qui peut être grand.

Toutes les solutions nécessitent presque la même quantité de mémoire pour stocker un élément dans la table de routage, mais c'est la solution 4 qui offre les conditions suffisantes pour le codage des adresses.

4 Conclusion et perspectives

Toutes les solutions proposées, sauf l'approche à base de deux filtres de Bloom, se sont avérées utiles dans notre scénario. Cependant la solution 4, en mesure de supporter la plus grande quantité d'éléments dans la table de routage, est probablement l'option la plus souple. Nos futurs travaux visent au développement de techniques d'optimisation permettant de réduire les coûts de calcul et d'améliorer l'efficacité du relayage de paquets.

Références

[HP06] P. Hebden and A.R. Pearce. Data-Centric Routing using Bloom Filters in Wireless Sensor Networks. In *Intelligent Sensing and Information Processing, 2006*, pages 72 – 77, 2006.

[Mit02] Michael Mitzenmacher. Compressed bloom filters. In *IEEE/ACM Transactions on Networking (TON)*, volume 10, pages 604 – 612, 2002.

[TRL12] S. Tarkoma, C.E. Rothenberg, and E. Lagerspetz. Theory and Practice of Bloom Filters for Distributed Systems. In *IEEE Communications Surveys and Tutorials*, volume 14, pages 131 – 155, 2012.

[XL06] Jun Xu Xiuqi Li, Jie Wu. Hint-Based Routing in WSNs Using Scope Decay Bloom Filters. In *Networking, Architecture, and Storages (IWNAS'06)*, 2006.